

# Class `string` and String Stream Processing: A Deeper Look

# 21

## Objectives

In this chapter you'll:

- Manipulate `string` objects.
- Determine `string` characteristics.
- Find, replace and insert characters in `strings`.
- Convert `string` objects to pointer-based strings and vice versa.
- Use `string` iterators.
- Perform input from and output to `strings` in memory.
- Use C++11 numeric conversion functions.



## 2 Chapter 21 Class string and String Stream Processing: A Deeper Look

### Self-Review Exercises

**21.1** Fill in the blanks in each of the following:

a) Header \_\_\_\_\_ must be included for class `string`.

ANS: `<string>`.

b) Class `string` belongs to the \_\_\_\_\_ namespace.

ANS: `std`.

c) Function \_\_\_\_\_ deletes characters from a `string`.

ANS: `erase`.

d) Function \_\_\_\_\_ finds the first occurrence of one of several characters from a `string`.

ANS: `find_first_of`.

**21.2** State which of the following statements are *true* and which are *false*. If a statement is *false*, explain why.

a) Concatenation of `string` objects can be performed with the addition assignment operator, `+=`.

ANS: True.

b) Characters within a `string` begin at index 0.

ANS: True.

c) The assignment operator, `=`, copies a `string`.

ANS: True.

d) A pointer-based `string` is a `string` object.

ANS: False. A `string` is an object that provides many different services. A pointer-based `string` does not provide any services. Pointer-based `strings` are null terminated; `strings` are not necessarily null terminated. Pointer-based `strings` are pointers and `strings` are objects.

**21.3** Find the error(s) in each of the following, and explain how to correct it (them):

a) 

```
string string1{28}; // construct string1
string string2{'z'}; // construct string2
```

ANS: Constructors for class `string` do not exist for integer and character arguments. Other valid constructors should be used—converting the arguments to `strings` if need be.

b) 

```
// assume std namespace is known
const char* ptr{name.data()}; // name is "joe bob"
ptr[3] = '-';
cout << ptr << endl;
```

ANS: Function `data` does not add a null terminator. Also, the code attempts to modify a `const char`. Replace all of the lines with the code:

```
cout << name.substr(0, 3) + "-" + name.substr(4) << endl;
```

### Exercises

*NOTE: Solutions to the programming exercises are located in the `ch21solutions` folder.*

**21.4** (Fill in the Blanks) Fill in the blanks in each of the following:

a) Class `string` member function \_\_\_\_\_ converts a `string` to a pointer-based `string`.

ANS: `c_str`

b) Class `string` member function \_\_\_\_\_ is used for assignment.

ANS: `assign`

c) \_\_\_\_\_ is the return type of function `rbegin`.

ANS: `string::reverse_iterator`

d) Class `string` member function \_\_\_\_\_ is used to retrieve a substring.

ANS: `substr`

**21.5** (*True or False*) State which of the following statements are *true* and which are *false*. If a statement is *false*, explain why.

- a) strings are always null terminated.

ANS: False. strings are not necessarily null terminated.

- b) Class `string` member function `max_size` returns the maximum size for a `string`.

ANS: True.

- c) Class `string` member function `at` can throw an `out_of_range` exception.

ANS: True.

- d) Class `string` member function `begin` returns an iterator.

ANS: True (it returns a `string::iterator`).

**21.6** (*Find Code Errors*) Find any errors in the following and explain how to correct them:

- a) `std::cout << s.data() << std::endl; // s is "hello"`

ANS: The array returned by `data` is not null terminated.

- b) `erase(s.rfind("x"), 1); // s is "xenon"`

ANS: Function `erase` is a `string` class member function (i.e., `erase` must be called by an object of type `string`).

- c) 

```
string& foo() {
    string s("Hello");
    ... // other statements
    return;
}
```

ANS: A value is not being returned from the function (i.e., the `return` statement should be `return s;`). The return type should be `string` not `string&`—reference returns are dangerous.